CS304 Practice Final

Worth: 30% (or 100% if you score better than your combined average on assignments + midterm)

Instructions: 3 hours, pen and paper only. This practice exam is slightly longer than the real final.

Q1: give the full set notation definition of Θ notation. It begins like "*we define by* $\Theta(n)$ *the set of all functions* …" and draw the accompanying plot showing $f(n)$ and $g(n)$.

Q2: do the same as Q1 but for O notation.

Q3: show that $\frac{1}{2}n^2 - 3n = \Theta(n^2)$ by determining positive constants c_1, c_2 and n_0 according to your definition from Q1.

Q4: write the following function in O notation: $6 \cdot \log(n) \cdot n + 36 \cdot n - 3n^2$

Q5: what is the time complexity in O notation of the following pseudocode?

```
binarySearch(arr, x, low, high)
    repeat till low = high
           mid = (low + high)/2
               if (x == arr[mid])
               return mid

               else if (x > arr[mid]) // x is on the right side
                   low = mid + 1

               else                  // x is on the left side
                   high = mid - 1
```

Q6: write a c++ function with $O(n + nlogn)$ time complexity (skeleton below). The function doesn't need to do anything useful, it just needs to run in $O(n + logn)$ time.

```
void fun(int n){…}
```

Q7: below is the code for insertion sort with an accompanying main. Show what this program will print when it runs.

```cpp
template <typename T>
void insertion_sort(std::vector<T>& v) {
    for (int j = 1; j < v.size(); j++) {
        auto key = std::move(v[j]);
        int i = j - 1;
        while (i > -1 && key < v[i]) {
            v[i + 1] = std::move(v[i]);
            i--;
        }
        v[i + 1] = std::move(key);
        std::cout << v[0] << std::endl;
    }
}

int main()
{
    std::vector<int> input = { 4,2,5,1 };
    insertion_sort(input);
}
```

Q8: below is the code for quick sort. Modify this code so it runs in $O(n \log n)$ time on sorted input.

```
int partition(std::vector<int>& arr, int p, int r)
{
        int pivot = arr[r];
        int i = p - 1;
        for (int j = p; j < r; ++j)
                if (arr[j] <= pivot)
                {
                        ++i;
                        std::swap(arr[i], arr[j]);
                }
        std::swap(arr[i + 1], arr[r]);

        return i + 1;
}

void quicksort(std::vector<int>& arr, int p, int r)
{
        if (p < r)
        {
                int q = (p + r) / 2;
                quicksort(arr, p, q - 1);
                quicksort(arr, q + 1, r);
        }
}
```

Q9: write the copy constructor and move assignment operator for the following class:

```
template <typename T>
class LargeTypeRaw {
public:
    // Default Constructor
    explicit LargeTypeRaw(int size = 10)
        : size{ size }, arr{ new T[size] }
    {}

    // Destructor
    ~LargeTypeRaw() {
        delete[] arr;
    }
    bool operator<(const LargeTypeRaw& rhs) {
        return (size < rhs.get_size());
    }

    int get_size() const {
        return size;
    }

private:
    int size;
    T* arr;
};
```

Q10: fill in the time complexity in O notation for the following sorting algorithms on the following input types:

Algorithm/input type	Sorted array	Random array	Array of all zeros
Insertion sort			
Selection sort			
Mergesort			
Quicksort			
Radix sort			
Treesort (using BST)			
Heapsort			

Q11: write c++ code to insert an element into the front of a singly-linked list (definition below)

```
//Definition for singly-linked list.
struct ListNode
{
    int val;
    ListNode* next;
    ListNode() : val(0), next(nullptr) { }
    ListNode(int x) : val(x), next(nullptr) { }
    ListNode(int x, ListNode* next) : val(x), next(next) { }
};
```

Q12: write c++ code to remove an element from a doubly-linked list (skeleton below, position is given)

```
void remove(Node* position){…}
```

Q13: insert the following elements into a binary search tree. Show the tree after each insertion.

```
Elems = [1,2,3,4,10,9,8,7,-1,2.5]
```

Q14: insert the following elements into a binary min heap. Show the heap after each insertion.

```
Elems = [5,4,0,2,1,6,3]
```

Q15: remove the following elements from the final tree in Q13, show the tree after each deletion.

```
Elems = [3,1,10]
```

Q16: show the heap from Q14 after 3 calls to deleteMin (show the heap after each call)

Q17: repeat Q13 but for an AVL tree.

Q18: remove the following elements from the final AVL tree in Q17, show the tree after each removal.

```
Elems = [1,2,3,4]
```

Q19: convert the following infix expression to postfix, and then evaluate the postfix expression

```
Expr = 3*(1+2)-(5+2)*7
```

Q20: convert the following infix expression into an expression tree, and then produce the prefix and postfix versions of the expression by using the preorder and postorder traversals

```
Expr = (1+2)*(3+4)*(5+6)-7
```

Q21: using the following hash function, insert the sequence A=[60,21,11,70,82] into a hash table with initial capacity 10. Use linear probing as the collision resolution method.

```
Hash function = key % TableSize
```

Q22: which operation is faster on average in a heap: `insert` or `deleteMin`?

Q23: write the C++ `rotateWithLeftChild(AvlNode*& k1)` code for an AVL tree

Q24: write the C++ `remove(const T& x, BinaryNode*& t)` for a BST

Leetcode problems (I will release a full list of potential questions soon, and choose 1 or 2 for the final)

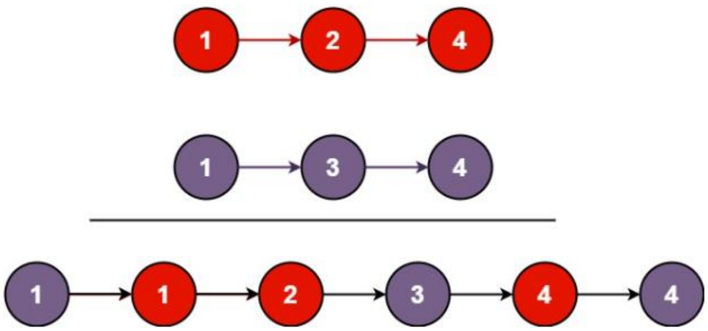
LQ1: merge two sorted lists

You are given the heads of two sorted linked lists `list1` and `list2`.

Merge the two lists in a one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list.*

Example 1:




```
Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]
```

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {

    }
};
```

LQ2: two-sum (your solution must run in $O(n)$ time, where n is the size of nums

Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to* `target`.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
```

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {

    }
};
```

LQ3: Trapping rain water

Given `n` non-negative integers representing an elevation map where the width of each bar is `1`, compute how much water it can trap after raining.

Example 1:


```
Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.
```

```
class Solution {
public:
    int trap(vector<int>& height) {

    }
};
```